Social-Insect-Inspired Adaptive Task Allocation for Many-Core Systems

Matthew Rowlings, Andy M. Tyrrell, Martin A. Trefzer
Intelligent Systems Group, Department of Electronics, University of York, York, YO31 5DD, UK
Email: mr589@york.ac.uk, andy.tyrrell@york.ac.uk, martin.trefzer@york.ac.uk

Abstract—**Large social insect colonies require a wide range of important tasks to be undertaken to build and maintain the colony. Fortunately, in most nests there are many thousands of workers available to offer their assistance to ensure the expansion and survival of the colony. However, there is a crucial equilibrium between the number of workers performing each task that must not only be maintained but must also continuously adapt to sudden changes in environment and colony need. What is most fascinating is that social insects can sustain this balance without any centralised control and with colony members that have relatively little intelligence when considered on their own. Due to this simplicity and evident scalability it would seem that social insects have evolved an interesting scalable approach to task allocation that could be applied to very large many-core systems. To investigate this we have explored biological models of task allocation in ant colonies and applied this to a 36-core Network on Chip. This paper not only shows that effective decentralised task allocation is achieved, but also that such a scheme can adapt to faults and alter its behaviour to meet soft real-time constraints. Therefore, it is established that social insect inspired intelligence models offer a suitable metaphor and development direction for tackling the challenges introduced by dark silicon and in-field faults in a decentralised and adaptive fashion.**

I. Introduction

The complex, yet coordinated behaviours that social insect colonies have formed to allow colonies to thrive in a vast number of different and ever changing environments exhibit an impressive ability of providing adaptivity, scalability and survivability to a colony. Despite this, most of us would consider individual insects as rather simple beings, seemingly always preoccupied with a particular task with no obvious ambition in sight. The choice of task however is an important part of colony survival, species that build large colonies not only rely on a wide range of tasks to be completed but also require an appropriate allocation of colony members to tasks; indeed the colony may not survive if this balance is upset. An intriguing part of this task allocation coordination is that there is no hierarchy of command in a colony, no single or group of members are responsible for allocating tasks to colony members. Instead the task it performs is entirely upto each individual, and the crucial allocation of tasks *emerges* from the decentralised dynamics of the social interactions between colony members. The emergence of such a complex and important behaviour across tens of thousands of colony members is of significant interest to solving distributed problems, indeed task allocation is now an important part of designing *many-core systems*.

For many years the advancement of digital technology has steadily exploited the transistor performance and density gains anticipated by Moore's law [1] and Dennard scaling [2]. Modern technologies however have reached a fundamental transistor size where Dennard scaling starts to break down when scaling down further [3], resulting in a shift of focus away from single-core performance towards many-core systems [4]. This has been driven by fundamental power and thermal limitations caused by the breakdown of Dennard scaling, with the result that all transistors on the chip can no longer be switched at their maximum frequency. Thus the many-core approach is required to leverage as much as the chip's computational potential as possible by relying on speed-up from application parallelism on cores that run below their maximum frequency and from hardware accelerators to stay within the power and thermal budgets [4]. This limitation has been dubbed *Dark Silicon* and has been highlighted as a crucial problem for the semiconductor industries, with some predictions claiming that at an 8nm process over 50% of a chip may need to be powered off [3].

The processing elements of a many-core system are typically interconnected on a single device using a *Network on Chip (NoC)* [5] [6]; an interconnection scheme based on conventional networking where routers and channels are provided for communication between nodes. Many node topologies, interconnect options and constraint optimisations are possible [7], giving the hardware engineer a powerful platform for implementing systems that could be made dark silicon tolerant. This flexibility comes with its own engineering caveats however: the large number of parameters will require problem and system analysis to ensure that systems implemented within NoCs fit their requirements and may necessitate the need for heuristical approaches such as [8][9][10] to optimise the design space; this is especially relevant when we consider the extra thermal and power constraints imposed on the design by dark silicon. This approach also suffers as the analysis is done at design time and so cannot be adapted should the operating conditions or properties of the chip change during operation. However such flexibility is a key requirement for supporting future many-core system design paradigms such as dynamic task allocation, in field self-repair and autonomous online optimisation [11].

Thus we need networks that can self-organise and self-optimise without the need for offline analysis. To support good scalability and dynamic network reconfiguration, ideal networking design space optimisations should therefore not rely on global knowledge of the network layout; indeed if many-core systems do scale into the hundreds and thousands of cores as suggested, then any online analysis will be computationally infeasible particularly within embedded systems. Thus, we will need to take a decentralised approach to the Network on Chip that utilises information available at each node instead of relying on a global view of the entire network. We have started to address this approach in our previous work, whereby inspiration was taken from behaviours of social insect colonies and applied to the problem of routing in the Network on Chip [12][13]. It was shown that social insects are a suitable metaphor for many-core networking as their communication structures fit the decentralised model well; simple communications between members result in self-organising behaviours emerging when observed globally at colony level. This was achieved by combining information from simple *monitors* within the network interfaces of each node with a small amount of local intelligence at each node to enable routing decisions to be made in a fully decentralised manner; the emergent behaviour resulted in desirable adaptive qualities such as routing around network hot-spots and enabling fault tolerance by dynamically routing packets around faulty nodes.

Following on from this work, the investigation presented in this paper explores the application of a model of social insect behaviours to achieve dynamic task allocation on a many-core system. In this case the routing behaviour is fixed and each node decides which task it should be performing in a decentralised and autonomous way. Task allocation models of the social insects, with a focus on ants in this case, are first introduced followed by a description of how this was transformed into a form suitable for many-core implementation. We then present our investigations in Section IV, the results of which illustrate the effective adaptive behaviour of the model, with a further development to show how more elaborate adaptive behaviours are supported including experiments with fault tolerance and soft real-time applications. Further extensions to this intelligence model are then proposed to enable integration with modern hardware systems to demonstrate how social insect intelligence models are an adept yet scalable metaphor for enabling autonomous self-optimisation and self-repair of future many-core systems.

II. Intelligent Task Allocation

Task allocation is the process of mapping tasks to nodes within the many-core and is a part of the multi-objective design space optimisation required for mapping applications to NoCs. It is easily seen that the layout of tasks on the grid will impact the network traffic profile of the many-core and so co-optimisation is required to optimise both dimensions to the problem. This becomes even harder when factors such as thermal constraints are also considered and if adaptation to changes to the topology (e.g. fault handling) is also required then there are many different scenarios to be considered and analysed.

If we now consider ant colonies in a similar fashion, we realise that some parallels are easily made. Each ant in the colony has to decide what task it should be undertaking at any one point and, of particular interest, *no methods of global organisation or coordination of work exist in the colony*. Despite this, colonies exhibit complex collective behaviours that are essential for the colony to survive. A broad spectrum of tasks are required to be undertaken: ranging from feeding and rearing of the young brood, nest expansion and maintenance tasks, to scouting for and retrieving food from outside of the nest. Differences in task partitioning and allocation between ant species was explored in [14] where they considered the differences between highly social ant species and species that build colonies of only a small number of members. In general they found that as social complexity increases (larger colonies), individual complexity decreased. They argued that there is a decline in autonomy of individuals of larger colonies, meaning they are less likely to be able to function on their own. This was also explored in terms of what tasks the members undertake, in smaller colonies the members tend to be “generalists”: they are able to undertake all tasks regardless of factors such as age. The greater differentiation in larger colonies means that members may be more optimal to perform certain tasks and specialists start to emerge, to the point that they may not exhibit their full task repertoire during their lifetime. This is analogous to systems comprising of a few but high performance general purpose processors (multi-core) against systems utilising a very large number of specialist cores (GPU, many-core). Thus we consider species that have large nest population and some degree of specialisation between members; mapping well to a hardware accelerator or FPGA approach where node reconfiguration can happen but at a temporal cost.

Many biologists have studied the task allocation of social insects and a comprehensive review of different models is considered in [15]. They explore six classes of models: response threshold, integrated information transfer, self-reinforcement, foraging for work, social inhibition and network task allocation models. Each model differs in what information source is used by individuals to determine which task they should be undertaking and so a brief summary of each model is given:

1) Response Threshold: In this model the assumption is made that workers are exposed to task-specific stimuli (e.g. dirty chambers, untended larva, hunger) and each worker has an internal threshold that dictates whether an individual decides to undertake a task depending on if a task stimuli exceeds this threshold, with a default behaviour of a “rest state” i.e. doing no task. The thresholds can vary between individuals and when a worker starts a particular task before other workers (it may have a lower stimulus threshold) it also start reducing the task stimulus for other workers, providing a lot of negative feedback into the task allocation system.
2) Integrated Information Transfer: This is an extension to the prior threshold model, whereby social information

transfer is also integrated into the threshold. Workers could inform each other with information on what tasks they perceive need to be undertaken, yielding a more step-wise distribution of task allocation from the positive feedback nature of the social-communication.

3) Self-Reinforcement: In an attempt to model the occurrences of specialists and generalists in the colony, experience based models have been proposed. In such models the decision to undertake each task is considered a probability, a successful undertaking of a task increases the probability that this task is performed again whilst an unsuccessful task or a lack of opportunity to undertake it will reduce the probability of the task being performed. This results in a self-reinforced system and by adding a notion of "forgetting" it allows specialists to revert back to generalists should the balance of tasks in the colony change.
4) Foraging For Work: This model uses a production line analogy such that there are a series of tasks to be done, geometrically spread. On contact with a particular task an individual will perform this task until it is no longer required (see task stimuli in the Response Threshold model), at which point it will then roam the nest until a new task to be done is found. This model predicts temporal polyethism, for example newly hatched workers will start their lives in the brood chambers at the back of the nest and so will find brood tending tasks to perform. However once this reaches a critical limit of workers then there will be a time where no brood tasks need to be performed and they will slowly work their way to the front of the nest as they forage for new tasks.
5) Social Inhibition: Another explanation of temporal polyethism can be obtained by considering the effect of older workers as an inhibitor for younger workers taking up new tasks. If a number of foragers is lost then the number of mature workers in the nest is reduced, resulting in less inhibition of the potential tasks a young worker can perform and so making up for the loss of foragers. However such a model for task allocation assumes that all tasks decisions are polyethism and inhibition driven, which has been shown not to be the case with task stimuli.
6) Network Task Allocation Models: The final modelling method considers the interactions between workers and their environment as a series of differential-equations or network models. The resultant models show similarity to real colonies to an extent that it can be concluded that division of labour can be generated and maintained purely from the local information encountered by an individual worker.

Finally the authors conclude that these models should be considered "exploratory" and that in reality some hypotheses of these models will eventually be refined and merged to produce a final "explanatory" model (for example the merged threshold and reinforcement models in [16]).

This gives us a choice of models that we could use for task allocation on the many-core. As we are not interested in precisely modelling the actual interactions and stimuli of social insects, we are more interested in the network-based models. Indeed Gordon explores task allocation as an ad hoc, dynamical network in [17], and explores interactions between members as the foundation for task allocation with suggestions such as: "what matters to an ant is the pattern of interactions it experiences, rather than a particular message or signal transferred at each interaction". Indeed this could be analogous to a node monitoring the properties of packets it encounters (priority, rate, destination) rather than the actual data contained within the packet. In fact the parallel distributed model Gordon proposes in [18] could quite easily be adapted for many-core experiments. Simple threshold decision functions were used by each agent (member of the colony) to determine which of eight states the agent should currently be fulfilling. It was found that not only did this model exhibit several characteristics of colony dynamics, it also allowed perturbations in tasks to be introduced and the agents in the system would then adapt their states until it would eventually return to a normal, stable state.

This is exactly the type of adaptive behaviour that we want from our many-core and so we have adapted the model for implementation in each router in the NoC. We implemented a simple 5-port router (N, E, S, W and internal node) and assumed a method of communication between the router and the internal node that allows the router to inform the node which task it should currently be undertaking (this could be as simple as a few wires to communicate the value or could be more complex such as a special packet sent from the router to the node). The task switch process is shown by the four steps in Figure 1

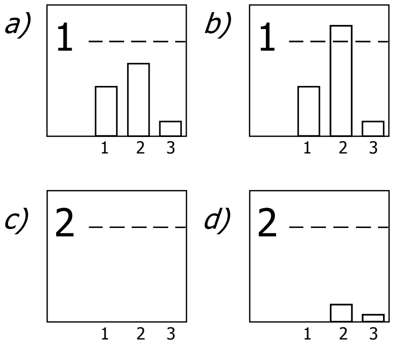

Fig. 1. Task switch decision algorithm for an example with three tasks. *a)* When a packet arrives at the router the router inspects what task the packet is destined for. The router then increments an internal counter of the destination task of the packet. *b)* If a task counter exceeds the *task switch threshold* then the decision is made to change (or maintain) to that task and the application node is informed. *c)* All of the counters are then reset *d)* The router then starts the task switch process again

III. Experimental Setup

The task allocation scheme introduced in Section II was experimented with by implementing the scheme in the routers of a high level NoC simulation. This was implemented using

TABLE I
APPLICATION MODEL SETTINGS. THESE DETERMINE WHEN A PACKET IS GENERATED, AS EXPLAINED IN SECTION III AND FOR EACH OF THE APPLICATION GRAPHS SHOWN IN FIGURE 2 AND 3

	Scenario 1				Scenario 2		
Task:	1	2	3		1	2	3
Ratio:	1	1	1		1	1	1
Rate:	10	0	0		10	0	0
Packets Required:	0	1	1		0	1	1
CPU Time:	1	1	1		1	10	1

System-C [19] to allow realistic simulation of NoC specific effects such as interconnect latencies and packet buffering. A 6x6 NoC was simulated with a grid topology such that all nodes have four cardinal neighbours aside from the nodes at the edges. Each interconnect between nodes has a buffer of 25 packets on the receiving end and it was simulated that it takes a packet 10μs to be routed between nodes.

As with our previous experiments ([12][13]), two application scenarios were used for the experiments: one representing a balanced application and the second an application bottleneck where the ratio of each tasks are not equal. The application graphs for these scenarios are shown in Figure 2 and 3. A difference from the previous experiments is that the packets are no longer generated at a fixed rate. Packets from Task 1 nodes are generated at a fixed rate of one every 10 μs, whilst Task 2 and 3 nodes do not send a packet out until they have received a packet with their task as the destination - this introduces causality into the model and is a more realistic processing stream. Each node also has a "CPU time", this is a period that the internal port cannot sink packets due to the attached node processing a previously sunk packet and so incoming packets are passed on. These settings are summarised in Table I

An issue with allowing nodes to change their processing task is that it will render preset routing tables invalid when a task switch happens. As dynamic updating of routing tables is a research problem in itself we shall not approach this problem in this paper. Our approach however should adapt to an non-optimal routing scheme to provide better performance, many packets of the wrong task being sent to a particular node should result in that node deciding to switch its task to fit this routing pattern. For this reason it is decided to preset each router with a random routing table for each experiment, we then expect the task allocation scheme to adapt to this non-optimal routing pattern. Of interest is that this makes the system susceptible to cyclic livelock situations where the packets are routed in a circle, never reaching a suitable destination node. This will in turn create a large response in the routers handling these packets, causing them to decide to initiate a task switch to the task required by the cyclic packets and so breaking the cyclic behaviour.

All routers use a simple Round Robin approach to choosing which port to service i.e. ports N, E, S, W, I are serviced in turn continuously. As covered in our earlier work, Round Robin is a simple, decentralised yet fair strategy, albeit limited in intelligent capabilities. For the experiments we have also re-introduced the concept of node "hunger" from our previous work. When a node is busy processing data it sets a flag that tells its immediate neighbours that it currently does not want to receive packets, a neighbouring node could then decide to send its data to another node to exploit task parallelism in the network.

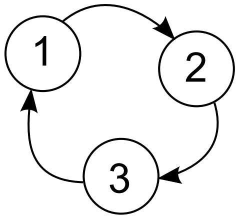

Fig. 2. Application graph for the first scenario. This represents a simple balanced processing application resulting in balanced traffic profile across the network, perturbed only by the network topology.

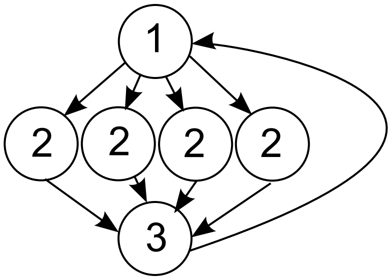

Fig. 3. Application graph for the second scenario. A data pipeline with a parallel stage is represented here whereby there are four times as many task two nodes as task one or three nodes. This can represent a typical many-core streaming application with a stage that is massively parallel, all data rates are kept the same as in the previous, balanced application graph shown in Figure 2 aside from CPU time for Task 2 nodes as shown in Table I. This scenario effectively increases the load on Task 2 nodes.

IV. INVESTIGATION AND RESULTS

A series of experiments using the simulated many-core system described in the previous section are presented here. Firstly the performance of the task allocation scheme for both application scenarios is analysed. Then we explore the fault tolerance capabilities of such a scheme - it is easily seen that if a node should become faulty then the task allocation across the network will no longer be balanced and the network should adapt to the faults. Finally an experiment exploring the addition of an extra monitor to the intelligent task allocation shows how we can advance this model towards real-time applications through the addition of deadlines.

Every experiment records the average time taken for a packet to traverse from its source node to its target node and its packets are continuously dispatched until 5000 packets have been sent throughout the network. This experiment is repeated 100 times with different randomly generated routing tables and starting topology in each run. This allows a statistical outline of the performance across many variations in network node topology to be measured, capturing the mean performance as well as the worst and best case outliers.

A. Task Allocation Performance

Figures 5 and 6 show the distribution of the average packet latency of each task for each application scenario and across 100 runs of the experiment. For the sake of comparison, the simulation was exploited to allow updating of all of the router's routing tables with new optimum paths when a task switch happens. This is the fourth scheme and allows us to anticipate the expected performance benefit that a dynamic routing update could provide. In the case with no task switching enabled, the routers are preloaded with optimal routing tables whilst in the task switching cases they are preloaded with random directions. This leads to a large spread in the task switching results in both application scenarios, however the medians are not drastically worse in the first application implying that a near comparable performance can be recovered despite the poor routing tables. Indeed when the latencies over time are considered in Figure 4 we can clearly see the adaptive task allocation at work, initially very poor performance is seen but this gets better as the dynamics of the network adapt until the performance plateaus; in this case with a ≈50% improvement over the random starting point.

When we enable the hungry flag we get a much better spread of packets across the network and hence a much tighter distribution. In fact this spread is very important for dynamic task allocation as if the fixed routing tables are followed then there are some nodes that will never get packets sent to them and so will not be able to adapt to the task requirements of the network. This is also clearly seen in Figure 4, the descent to good performance is much faster and the average much more stable than compared to task switching with no hunger flag. Some perturbations are seen in this plot but this is to be expected in an adaptive system and the system consistently returns to a stable state of improved performance. The extra optimality offered by updating the routing table can also be seen in Figure 4 but it is not exceptionally better for this run, especially considering the overhead that such an update would take to calculate in a real system.

B. Fault Injection Experiment

An exciting prospect of dynamic task allocation is the autonomous recovery of performance under faulty scenarios. If we consider the second application scenario then it can be seen that a loss of a Task 3 node would dramatically increase the workload of the remaining Task 3 nodes as there are many Task 2 nodes to process packets from. In our network this will alter the balance of packets in the network such that the overall rate of packets that Task 3 nodes send out will drop, potentially causing the thresholds in other nodes to initiate a task switch.

To experiment with this we undertook an extreme fault case whereby after 500ms of execution *all* of the Task 3 nodes are disabled. Their routers can still route packets but their attached node will not produce or sink packets. Packets destined for Task 3 nodes will therefore be constantly routed around the network until a working node(s) decides to perform a task switch. Ultimately the performance of other tasks will have to be sacrificed to fulfil this lack of Task 3 ability, but due to the causal nature of our test application it can be seen that by reducing the number of Task 1 nodes (the only node that produces packets at a fixed rate) the overall load on the network will decrease. Eventually this allows the network to return to a similar load to before the fault, albeit with a decrease in the total amount of processing being achieved due to loss of capacity from the faulty nodes. In an autonomous system this is usually a highly desirable failure mode as it allows graceful degradation of full system performance rather than system failure.

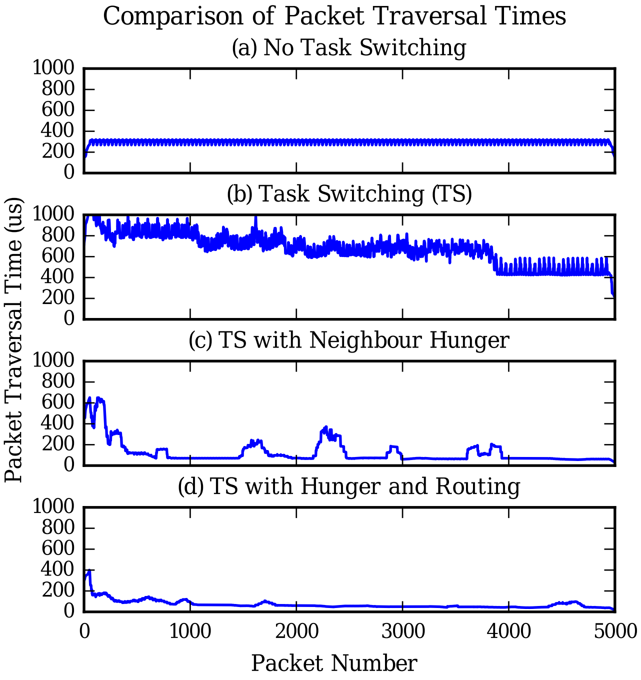

Fig. 4. Average packet traversal time represented in the temporal domain. A moving average was applied across the raw latencies of each packet in one of the 100 runs that make up the box plots. The same seed is used for each run of the different schemes and so the random starting topology and preloaded random routing tables are consistent across each scheme in this graph. This shows how stable each scheme is and the time required for the task switching behaviour to "settle down".

This fault injection and recovery sequence can be seen in Figure 7. The effect of the fault on the system is clearly seen with varying recovery times between the schemes as the time taken for nodes to switch task elapses. Indeed it could even be the case that an over compensation happens, whereby too many nodes switch task to recover the functionality; possibly causing the instability seen for the task switch case (graph a.). The optimum routing update case is clearly the most stable and quick to recover, this is probably as the information about a recovered Task 3 node is distributed the fastest via the routing table update instead of local inference from network traffic as with the other two schemes.

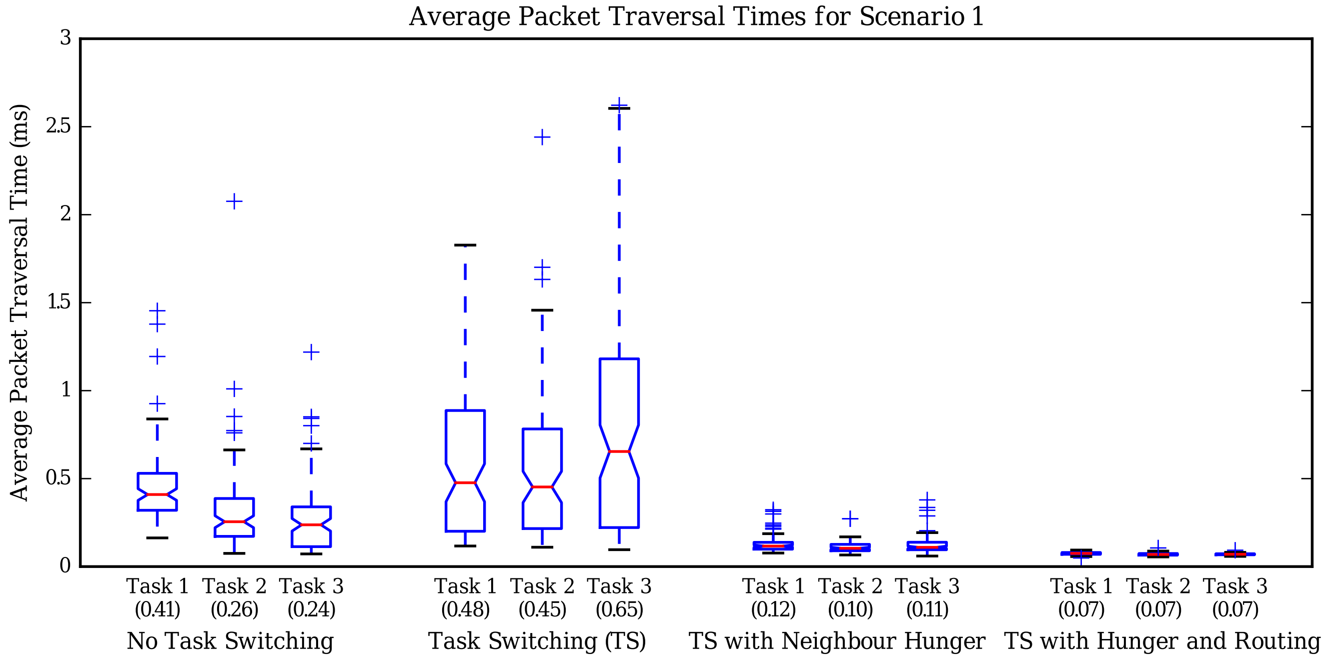

Fig. 5. Average packet traversal times for the first application scenario, with the median traversal time given in brackets on the x-axis. Despite the optimal routing tables for the no task switching case there are still some extreme outliers present, showing that some task allocation topologies are inherently inefficient and so task switching will be required to optimise this. With just task switching enabled we see a larger spread of average latencies when compared with no task switching; however the task switching experiment is loaded with a random routing table. Thus early packets will have high latencies until the network has adapted, see Figure 4 for an example of this effect. A "god mode" is used to update the routing tables for the final set, this is purely for comparison and shows that even with no routing table updates (the third scheme), the task switching with hungry flag performs well.

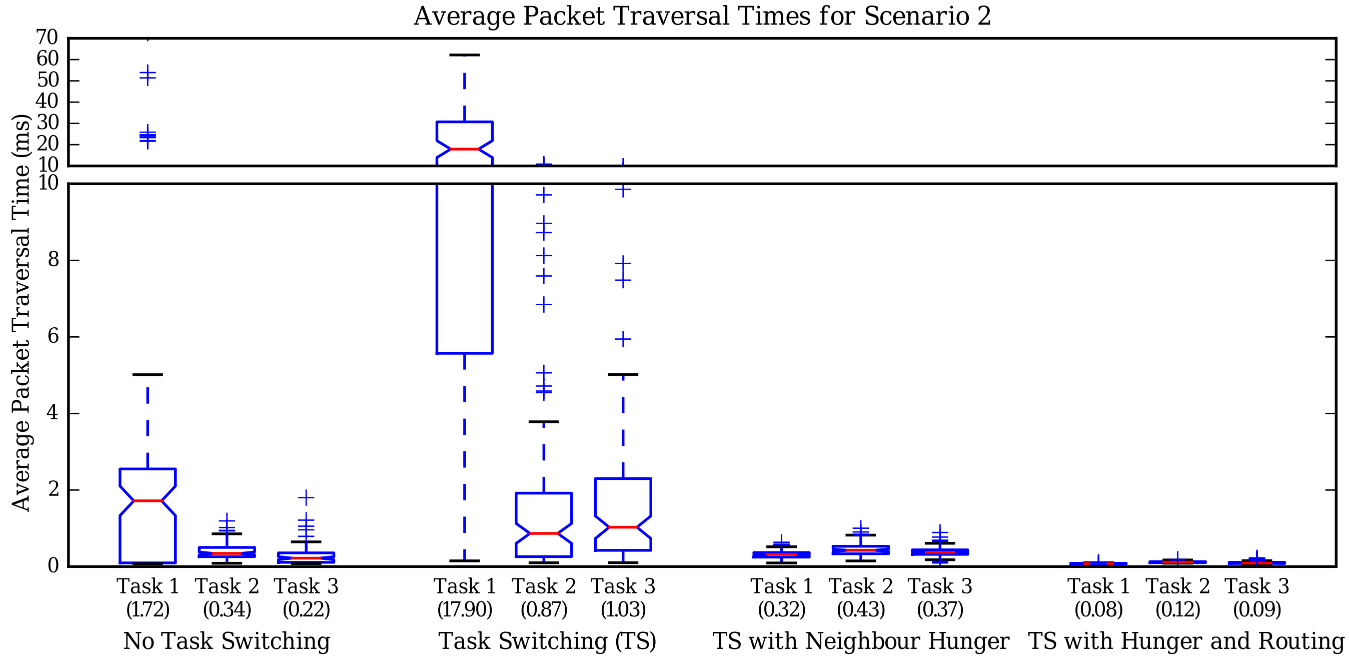

Fig. 6. Average packet traversal times for the second application scenario. Due to the large spread of task 1 latencies in the task switching case we have had to provide a second scaled axis. The hunger flag provides a much balanced spread of data across the network, this is clearly shown by the severely reduced latencies for task 1 packets - showing that the network adapts to this processing bottleneck. Once again the median traversal times are given in brackets on the x-axis.

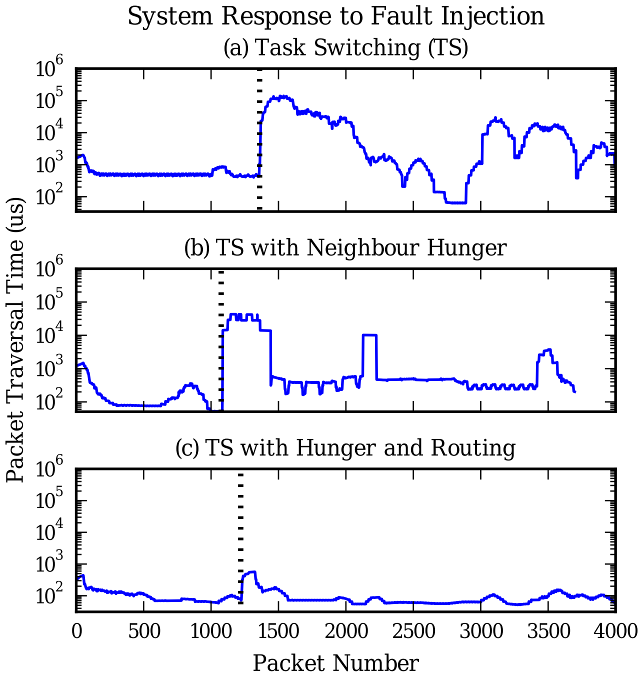

Fig. 7. Each of these graphs show the response of the system to a catastrophic fault under the three task allocation schemes (without task switching it would not be possible to recover, hence the no task switching case is not shown). The first application scenario is run and then all task three nodes were failed at the point shown on the graph as a dotted line. The logarithmic scale shows the severity of the extreme fault case, followed by varying speeds of recovery and some minor instabilities still present post-recovery.

C. *Towards Real-Time Applications*

Many autonomous embedded systems are implemented in environments where a calculation will be required to be done within a given time, in some cases information being late could be more detrimental than a fault in the system. We can see many time dependant behaviours in Nature in the form of *startle behaviour*: if a predator is attacking then you may well need to act faster than the predator to survive. At a neuronal level this had led to many organisms evolving startle reactions that can even shortcut the usual nerve-brain-muscle pathways to ensure a quick reaction [20]. Taking inspiration from this our third investigation added the requirement that *all packets should finish with 100μs*, that is be routed from source to destination task within 100μs. An extension was added to the router that read the created timestamp contained in the packet and this was then compared to the current time. If more than 100μs had elapsed since the packet was created then the node immediately undertook a task switch to be able to process the packet. This is not a temporary task switch - the counters for all tasks are cleared as with a normal task switch and the node continues processing with the task it had urgently switched to until it may decide to switch again. Indeed with such a deadline approach to packets then the concern is how optimal the network is at meeting deadlines as opposed to processing packets as fast as possible.

As can be seen in Figure 8, this simple modification is highly effective. The spread of packets with just task switching enabled is much smaller and the medians are very close to the desired 100μs. Once the hungry flag is enabled the entire distribution is within the deadline and the difference of advantage offered by the routing update is minimal. This shows that it is possible to merge the intelligence model with other decision pathways and is a promising indication that the resilience of bio-inspired adaptive systems may be integrated with the stringent engineering requirements of more critical systems.

V. Conclusions

We have demonstrated here how task allocation in Social Insects can be applied effectively to many-core systems to achieve adaptive task allocation across the many-core. The decentralised nature of this model is highly desirable and required as many-core systems start to scale into the hundreds and thousands of nodes; rendering traditional design space exploration approaches infeasible. We have shown how a simple behaviour at each node results in a model that intrinsically copes well with faults, allowing autonomous systems to gracefully bring back functionality from even catastrophic failure situations. This has been furthered by introducing more complexity into the intelligence, a panic-like reaction allows approaching deadlines to be serviced by exploiting the task switch to ensure packet deadlines are met. Our results indicate that our approach will be an appropriate mechanism to overcoming the problems introduced by Dark Silicon and other issues of extremely large scale integration that require adaptive solutions.

VI. Further Work

As we saw in Section II there are many more models of social insect task allocation and although we shall not implement each one in turn, we could take mechanisms from other models and refine them for use with the implemented model. For example the Self-Reinforcement model would allow nodes to discover if they had a particular advantage undertaking certain tasks over others (use of a hardware accelerator for example) and could also offer a autonomous approach to dynamically managing the balance of “specialist” and “generalist” nodes in the network depending on the current application profile. Indeed it is generally accepted that polymorphism (whereby members more physically suited to a particular task are more likely to undertake it) is a key aspect of division of labour in social insect colonies and so we are keen to exploit this model as it maps so well to specialist hardware accelerators that we could have in our many-core.

From a systems perspective we have made some simplifications in this model such as assuming an instantaneous task switch and that all nodes can perform all tasks. When integrating with an actual system we shall have to consider such costs and constraints as part of the decision making process, possibly through extending the monitors in the system and integrating these into the intelligence model. We would then look to implement these experiments within the RISA

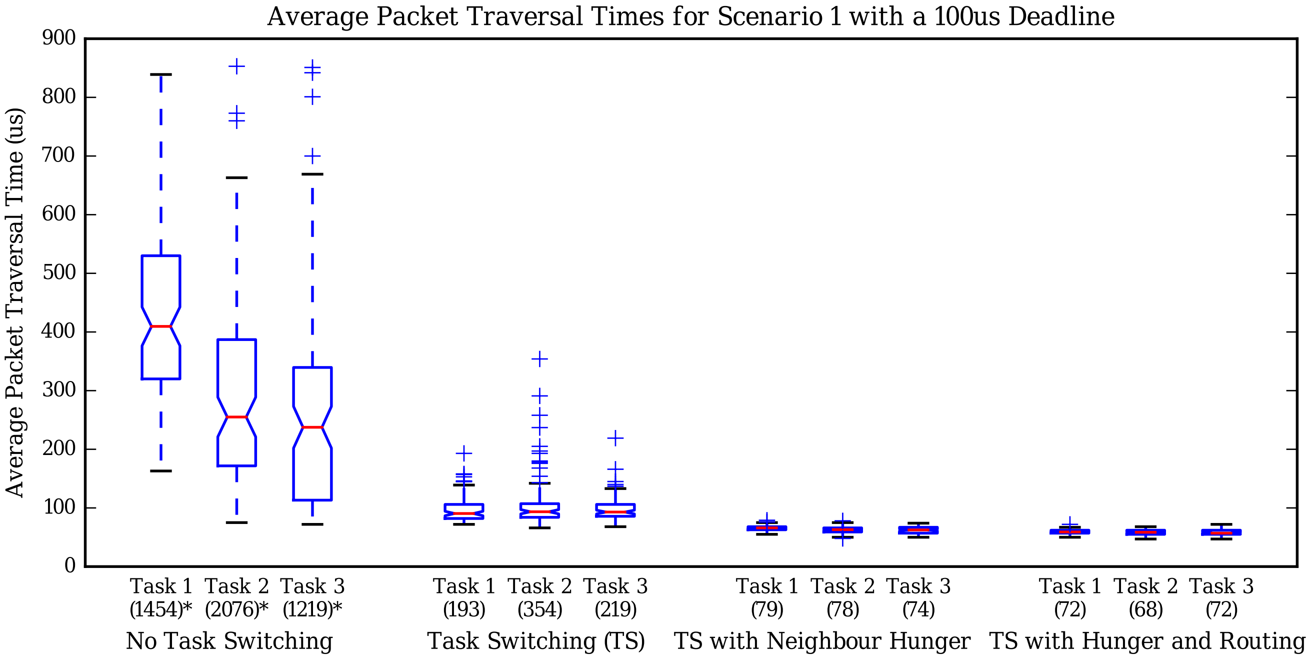

Fig. 8. Results of the first application scenario with a deadline of 100μs applied. In this plot the upper bound of the average latency for the runs is given in brackets. It is shown that in terms of meeting the deadline the Hunger flag with task switching is of comparable performance to the same scheme with optimal routing updates enabled. This is of interest as it shows the flexibility of the network in this case is as important as optimal routing. Indeed this is even shown in the first half of the plot as the task switching scheme has managed to provide a far tighter distribution that the no task switching (but optimal routing) scheme. Thus we have captured some of the flexibility gains that social insect colonies have formed to exploit. **The upper outliers of the No Task Switching case have been omitted from the plot to give better clarity to the results of the other schemes with task switching enabled.*

many-core comprising of 36 nodes as we have done with previous investigations [13].

Acknowledgements

This work was supported by funding from the Department of Electronics and an EPSRC DTA award.

References

[1] G. Moore, “Cramming more components onto integrated circuits,” *Proceedings of the IEEE*, vol. 86, no. 1, 1998.

[2] R. Dennard and V. Rideout, “Design of ion-implanted MOSFET’s with very small physical dimensions,” *Solid-State Circuits, IEEE Journal of*, vol. 9, no. 5, pp. 256–268, 1974.

[3] H. Esmaeilzadeh and E. Blem, “Dark silicon and the end of multicore scaling,” in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011, pp. 365–376.

[4] G. Venkatesh and J. Sampson, “Conservation cores: reducing the energy of mature computations,” in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, 2010, pp. 205–218.

[5] L. Benini and G. D. Micheli, “Networks on chips–a new SoC paradigm,” *Computer*, 2002.

[6] A. Hemani, A. Jantsch, S. Kumar, and A. Postula, “Network on chip: An architecture for billion transistor era,” in *IEEE NorChip Conference*, 2000.

[7] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, “Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3–21, jan 2009.

[8] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, “On-chip communication architecture exploration,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 23–es, aug 2007.

[9] K. Srinivasan, K. Chatha, and G. Konjevod, “Linear programming based techniques for synthesis of network-on-chip architectures,” in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* IEEE, 2004, pp. 422–429.

[10] U. Ogras and R. Marculescu, “Energy- and Performance-Driven NoC Communication Architecture Synthesis Using a Decomposition Approach,” in *Design, Automation and Test in Europe.* IEEE, 2005, pp. 352–357.

[11] G. Tempesti, “Graceful Design,” *International Innovation Issue 140*, pp. 76 – 78, 2014.

[12] M. Rowlings, A. Tyrrell, and M. Trefzer, “Social-Insect-Inspired Networking for Autonomous Load Optimisation,” *Procedia CIRP*, vol. 38, pp. 259–264, 2015.

[13] ——, “Social-Insect-Inspired Networking for Autonomous Fault Tolerance,” in *2015 IEEE Symposium Series on Computational Intelligence*. IEEE, dec 2015, pp. 1198–1205.

[14] C. Anderson and D. McShea, “Individual versus social complexity, with particular reference to ant colonies,” *Biological Reviews (of the Cambridge Philosophical Society)*, pp. 211–237, 2001.

[15] S. Beshers and J. Fewell, “Models of division of labor in social insects,” *Annual review of entomology*, 2001.

[16] G. Theraulaz, “Response threshold reinforcements and division of labour in insect societies,” in *Proceedings of the Royal Society B: Biological Sciences*, 1998, pp. 327–332.

[17] D. Gordon, “The organization of work in social insect colonies,” *Nature*, 1996.

[18] D. Gordon, B. Goodwin, and L. Trainor, “A parallel distributed model of the behaviour of ant colonies,” *Journal of theoretical Biology*, 1992.

[19] IEEE, “IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005),” pp. 1–638, 2012.

[20] P. Simmons and D. Young, *Nerve cells and animal behaviour*, 3rd ed. Cambridge University Press, 2010.